ODYSSE: A routing Protocol for Wireless Sensor Networks

Ichrak Amdouni, Cédric Adjih, Nadjib Aitsaadi, Paul Mühlethaler

ODYSSE: A routing Protocol for Wireless Sensor Networks

Ichrak Amdouni, Cédric Adjih, Nadjib AitSaadi, Paul Muhlethaler

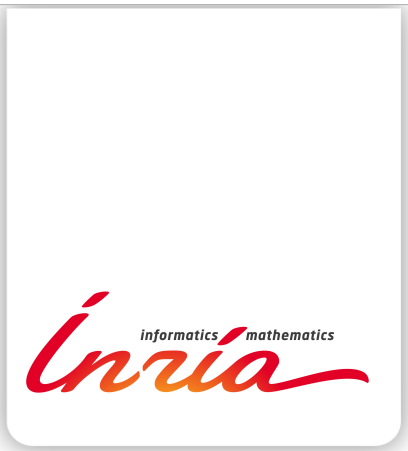
RESEARCH REPORT

N° 8873

February 2016

Project-Teams INFINE, Eva

ISSN 0249-6399 ISRN INRIA/RR--8873--FR+ENG



ODYSSE: A routing Protocol for Wireless Sensor Networks

Ichrak Amdouni*, Cédric Adjih, Nadjib AitSaadi*, Paul Muhlethaler

Project-Teams INFINE, Eva

Research Report n° 8873 — February 2016 — 33 pages

Abstract: In this article, we propose, design, model and experiment an energy efficient protocol for Wireless Sensor Networks (WSNs), ODYSSE ("Opportunistic Duty-cYcle based routing protocol for wirelesS Sensor networks"). It combines three main elements: the first one is *duty-cycling*, where nodes alternate between active and sleep states, and is a classical but effective method to save energy. The second one is *opportunistic routing* where relaying (routing) is not rigidly fixed: at each hop, any node closer to the destination might become the relay. This requires less node synchronization, allows for path diversity and load balancing. The third one, is *source coding* (with LDPC, Low-Density Parity-Check codes). With uncoordinated duty-cycling as a starting point, the three techniques fit perfectly, yielding a robust low complexity protocol for highly constrained nodes. Modeling the average waiting delay of forwarders, we also show that simple relay selection strategies are effective. We focused on two heterogeneous scenarios: the most challenging scenario of bulk-transmission (of still images), and one of the most classical WSNs applications, i.e infrequent events reporting. Using a testbed of 45 Arduino nodes communicating with IEEE 802.15.4 (XBee) within the large scale platform FIT IoT-LAB, we implemented and extensively studied the behavior and performance of the protocol.

Key-words: wireless sensor networks, duty cycle, opportunistic, energy efficieny, LDPC, source coding, experimentation, Arduino, FIT IoT-LAB

* LiSSi, Univ. of Paris-Est Creteil Val de Marne (UPEC), Vitry-sur-Seine, France

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

ODYSSE: Un protocole de routage pour les réseaux de capteurs

Résumé : Nous proposons ODYSSE, un protocole de routage opportuniste basé sur le duty-cycle pour les réseaux de capteurs sans fil. ODYSSE comprend trois composants: (1) *un mécanisme de duty-cycle* qui permet aux noeuds d'alterner entre l'état actif et l'état de veille pour économiser l'énergie, (2) *un routage opportuniste* où les routes ne sont pas fixées à priori: à chaque saut, le routeur choisi parmi des voisins réveillés, un relais qui soit plus proche de la destination. Ainsi, moins de surcoût est généré, les paquets passent par plusieurs chemins, et la charge est équilibrée; (3) *une technique de codage par la source* basée sur les codes LDPC ("Low-Density Parity-Check codes") pour améliorer la fiabilité des transmissions. En modélisant les délais moyens d'attente de relais, nous avons prouvé que des stratégies de sélection de relais simples sont efficaces en termes de délais. Nous avons mis l'accent sur deux scénarios hétérogènes: un des scénarios les plus contraignant dans les réseaux de capteurs: transmission continue de grands volumes de données, telles que des images, et un des scénrios les plus classiques dans les réseux de capteurs: supervision d'évènements rares. Nous avons réalisé des expérimentations ce protocole à l'aide d'une plateforme à base des noeuds Arduino et module radio XBee (IEE 802.15.4), intégrée dans la plateforme existente FIT IoT-LAB.

Mots-clés : réseaux de capteurs sans fil, opportuniste, économie d'énergie, LDPC, codage par la source, expérimentation, Arduino, FIT IoT-LAB

1 Introduction and Contributions

1.1 Background

The critical issue addressed in Wireless Sensor Networks often relates to energy efficiency. Solutions have been proposed for fixed patterns of communication, such as collecting low frequency sensor measurements [10]. However, another opposite communication pattern is the accumulation of data followed by bulk transmission (for instance for maintenance applications). A perfect illustration of higher volume requirements is given by the family of Wireless Multimedia Sensor Networks (WMSNs) [10, 11, 12]. WMSNs have shifted the focus from the typical scalar WSNs to networks which deliver multimedia traffic as video, audio, still image in addition to scalar data. The proliferation of these networks is explained by the increasing availability of commodity multimedia sensors like CMOS cameras, microphones and the significant progress in distributed signal processing and multimedia source coding techniques. The promising applications are numerous [10] . Multimedia content such as video or images can be used to monitor areas, public events, locate missing persons, record potentially relevant data for future use (in case of an accident for instance, authorities rely on recorded images to get further analysis for instance). From research point of view, supporting such applications holds many challenges, and in particular, the multimedia nature of the collected information may add application-specific QoS requirements.

Inspired by these WMSNs, we focus on a dynamic and adaptative routing protocol, which can both address low volume traffic and routing of large volumes of data. The constraints are energy efficiency while maintaining low end-to-end delays, objectives that are inherently in tension, and are satisfied by adaptability. In our work we face these challenges by proposing ODYSSE: an opportunistic routing based on duty cycle.

Applying duty cycling for sensors [1, 2] means alternating between active and sleep states while favoring the sleep state as much as possible. Obviously, such a design saves energy by minimizing the occurrence of energy-wasting situations such as idle listening, collisions and traffic overhead generation [3].

Duty cycling can be either synchronous or asynchronous. The first category implies clock synchronization and coordination between nodes to set their sleep and active states. An example of framework for synchronous duty cycling is the current work of the 6TiSCH working group from the IETF, where communications are coordinated from a slotted time division multiple access (TDMA) with frequency hopping [37]. In contrast, in asynchronous designs, sensor nodes wake up independently, and avoid the prior clock synchronization overhead at the expense of "online" sender-receiver coordination. It is easy to implement, can be performed locally and can easily adapt to all topology changes, and for this reason, ODYSSE, our proposal, relies on this type of synchronization. Namely, ODYSSE is based on sender-initiated coordination technique (like B-MAC [16], see section 5). One major concern of the duty cycle design is to meet delay requirements while increasing as much as possible the sleeping period of each sensor. Another common approach for energy efficiency in WSNs is opportunistic routing [4, 5]. The rationale behind it is to exploit the broadcast nature and to allow nodes that overhear a transmission and are closer to the destination to participate in forwarding the packet. Indeed, unlike deterministic routing which builds the specific route for each source-destination pair, opportunistic routing can instantly create route diversity to balance load by adaptively selecting forwarders at each intermediate hop. A side effect is that energy consumption may be spread more evenly over multiple paths, making also possible a forwarding strategy avoiding intermediate forwards with low battery levels. Moreover, latency is improved, along with capacity and fault tolerance.

Two methods are possible in opportunistic routing: multicast [6] where a packet is forwarded

by all nodes which receive it, or unicast [7, 9] where only one forwarder ensures packet forwarding. The drawback of the first category is the packet forwarding redundancy, while the second one faces the challenge of keeping an acceptable overhead for forwarder selection method. ODYSSE combines benefits of both categories. More than selecting the first awake neighbor, ODYSSE can wait for a specific duration for potential forwarders and can select the best one, based for instance on its residual energy or other metrics. We will also prove through theoretical modelling that selecting the first available forwarder is the optimal choice from delay perspective under some conditions. To further improve latency, a variant of ODYSSE is proposed: we will detail how *the duty cycle tuning* of this protocol is application aware; it is *adaptive* to the infrequent small data transmissions.

Finally, to enhance wireless transmissions reliability, either some feedback is implemented (which is still complex in duty cycling opportunistic routing), or when operating in open loop, a good design choice is forward error-correction methods. To keep the protocol simple, ODYSSE adopts this last open-loop approach based on LDPC coding.

1.2 Contributions

The major contributions of this work are:

- Design of ODYSSE: a distributed duty cycle based opportunistic routing protocol for WSNs. ODYSSE adjusts the duty cycle of routers to fit heterogeneous types of transmissions from multimedia applications characterised by bulk transmissions to infrequent event monitoring applications where data transmissions are rare (Section 2).

- Joint design of ODYSSE with packet erasure code, based on LDPC [27] codes. This code consists in sending redundant coded packets that allow the final destination to retrieve lost source packets, without consuming too much memory (RAM) (Section 2.7).

- Modeling of the average waiting time of a forwarder. One finding is of special interest: it can be more delay efficient for any node to select the first awake forwarder under some conditions. This is because the time needed to wait for a second forwarder can be sufficient for the first forwarder to progress more towards the final destination (Section 3).

- Deployment of a real 45 based arduino WSN testbed integrated in the existing large scale testbed FIT IoT-LAB. ODYSSE is implemented and evaluated on this real platform (Section 4).

2 ODYSSE Protocol

2.1 Network Architecture and Objectives

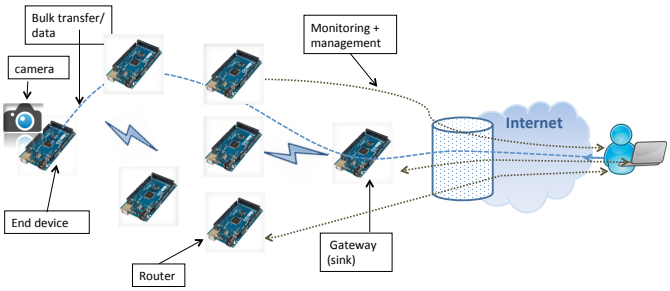


Figure 1: The network architecture.

We consider a network, depicted in Figure 1, which includes (1) one or more multimedia devices, called **source**(s), periodically generating bulk data (e.g. images); (2) router devices forwarding the multimedia traffic inside the mesh network; (3) gateway device collecting data from the mesh network. One example of targeted application is recording images from the monitored area for future usage (post-analysis of data in case of accident for instance [10]). We assume that all devices are battery powered, except for the gateway which has a permanent energy power (a classical setup in WSNs). The gateway has two interfaces: one with the mesh network enabling it to collect data generated at source nodes, and the second is linked to the user to allow the remote management of the network, monitoring the protocol, etc.

ODYSSE is responsible for routing data from sources to the gateway while **optimizing the lifetime of the network**. In addition, ODYSSE aims at **keeping low data transmission delays and overhead** since it deals with small capacity devices. In the following, basic ideas and detailed design of ODYSSE are provided.

2.2 Overview of the ODYSSE Protocol

To save energy, ODYSSE protocol makes **router nodes duty cycled**. Routers enter the sleep state asynchronously and randomly. This design avoids the synchronization overhead and the degradation of the system performance in case of clock drift. As a consequence, the wireless topology is dynamic and unpredictable which makes the application of the classical reactive and proactive routing not practical. Thus, to overcome this challenge, and to further reduce energy consumption, ODYSSE protocol adopts an **opportunistic routing** approach.

To keep low complexity forwarder research method, in ODYSSE, the forwarder is the neighbor offering a best compromise for the selected strategy depending on: **i) its residual energy, and ii) its distance relative to the gateway and iii): its quality of wireless link.** Hence, the multipath routes built can be optimized in terms of energy, reliability or distance.

To ensure routing progress towards the gateway and to avoid routing loops at each hop, a DAG (directed acyclic graph) rooted at the gateway is implicitly constructed and maintained by each node as its **'distance'** relative to this gateway. This distance is essentially the **number of hops** separating the node from the gateway but taking into account **the link quality** as a metric. In order to avoid pure *shortest path* routing, which would limit the width of the DAG (hence the alternate routes), longest links are eliminated. Our heuristic for characterizing long links

(in terms of distance) is based on **Received Signal Strength Indicator (RSSI)**, where the metric, "equivalent number of hops" for a link is a function of the RSSI, instead of just one hop. To determine their distance to the gateway, nodes broadcast a message initiated by the gateway, forming a logical tree. In this tree, each node has as a parent the node with the smallest distance from which the node has received such a message. Note that the formed logical tree is used for distance computation and not routing. The source node generates, in addition to the original data packets, **repair (redundant) packets** which are linear combinations of original ones. The repair packets are routed the same way as original packets. This data redundancy enables the final destination to decode the received packets in order to retrieve the lost packets. Our aim is to evaluate ODYSSE under real conditions; ODYSSE is implemented and experimented on a **real big testbed of Arduino** devices. ODYSSE is a distributed protocol. As routes are computed on the fly, nodes do not need to store topological information as in classical routing protocols. In the following sections, we detail its different operations.

2.3 Distance Computation

As indicated, a distance is essentially computed as a hop count. The variation is that it is a hop count using "not long" links (in terms of physical distance approximated through RSSI).

A simple link metric is sufficient, as opposed to some other WSNs cases, for the following reasons: classically, Estimated Transmission Count (ETX) [38] is used as a heuristic to minimize channel occupancy for routing (hence increasing throughput), while at the same time selecting links with higher delivery ratio. In our context, the primary limiting factor for throughput is duty cycling, not channel occupancy. Hence unnecessary packet retransmissions are less of a concern. Furthermore, in our opportunistic sender-initiated design, potential links are immediately tested at each transmission attempt, reducing the chances for worse links to be selected. Finally, data transmission failure (evidenced by lack of acknowledgement reception at MAC level, for several transmissions) triggers a restart of the forwarder selection procedure, not a permanent packet loss.

Therefore our objectives for the link metric are twofold: promote path diversity and optionally avoid spurious retransmissions in the spirit of minimizing energy consumption.

If pure shortest path routing were used, longer links would be selected (resulting in more likely packet losses), and it becomes more likely that only one next hop (with lower distance) exists. If only short links are considered for computing distance, path diversity is promoted as follows: it might be the case that a node has only one possible next hop with strictly inferior distance (again: reachable through a short link). However considering additionally "long" links, the node would be able to reach beyond that unique next hop , and is more likely to have additional potential forwarders. Finally, we do not want to totally exclude "long" links, as in some general topologies, it is possible that two parts of the networks are only connected through a "long" link. For these objectives, the RSSI is a good basic for computing link metric: for characterizing long links; it is also an indicator of packet loss . Of course, there is not a one-to-one deterministic mapping between distance and RSSI, but they are highly correlated (or between loss probability and RSSI). For instance, this is illustrated by measurements made in a similar context in [39]: in the fourth figure of [39], based on statistics from 19806 links, almost none of the links with distance ≤ 1 meter, have a RSSI lower than -70 dBm, while conversely, almost none of the links with distance ≥ 5 meters have a RSSI greater than -70 dBm. Similarly, sixth figure of [39] shows that almost all existing links have $> 95\%$ packet delivery ratio (PDR), except for losses interpreted by interference (80% PDR); data analysis shows that non-interference losses occur on links with worse average RSSI, e.g. RSSI ≤ -90 dBm. Hence, ODYSSE uses the following link metric, where `RSSI` is the RSSI at a receiver.

- If (`RSSI < RSSI_THRESHOLD`), then, `link-metric` $= 1 + \gamma$.
- Else, `link-metric` $= 1$.

The parameters `RSSI_THRESHOLD` and γ can be tuned experimentally. From this metric characterizing a given link, a classical distance vector protocol computes distances from the gateway. In the general case, the protocol is intended to be constantly updating distance, by piggybacking distance information on forwarder selection, and with expiration of older information, as done for instance in the protocol Destination-Sequenced Distance-Vector Routing (DSDV) [40]. However in our implementation and experiments, due to the great stability of the topology of the testbed, we use a prior initialization phase where the distances are computed for once. The protocol operates as follows: the gateway initiates the process to allow each node to estimate its distance denoted (`gateway-distance`) towards this gateway. This process is based on the broadcast of a message called `'Level'` message. This message is sent periodically by the gateway. The gateway initialises its distance to zero and broadcasts the `Level` message. Receiving this message from any node v, each node u performs the following steps:
1) The `gateway-distance` is potentially updated as:

- The `link-metric` between u and v is computed as described previously from the RSSI of the `'Level'` message.
- `candidate-distance` := `gateway-distance` of the transmitter v + `link-metric`
- If this node u has not yet computed its distance, or the candidate distance is less than the distance already computed, the node updates its distance accordingly: `gateway-distance` := `candidate-distance`.

2) The first step is iterated for a predefined period called `LEVEL_PERIOD` to receive potential `Level` messages from neighbors. After this period, in case the node has updated its distance, it generates a new `Level` message including its distance and broadcasts it. Otherwise, the message `Level` received is not repeated.
As a heuristic to compute more accurate distances, the retransmission delays are set so that approximatively, the absolute time of transmission of a `Level` message is proportional to the distance it advertizes. After these steps, nodes form a logical tree (actually, a DAG) defining their distances towards the gateway taking into account the RSSI values.

2.4 Forwarder Search and Selection

Routing is based on a **greedy approach**. In ODYSSE, forwarder selection is **sender-initiated**. When one node u has a data packet to transmit (either the source node or any router node), it broadcasts `Beacon` messages and awake neighbors answer by sending `Reply` messages.

More specifically, the `Beacon` message from u includes its distance to the gateway (denoted `gateway-distance`). Any awake node v receiving this beacon will proceed as follows:

1. If v is closer to the gateway than u and `RSSI < RSSI_THRESHOLD` then v sends a `Reply` message. This message includes among others: i) the identifier of the replier, ii), its distance with respect to the gateway, and iii) its residual energy.
2. Otherwise, the node v stays silent.

The initiator u will repeatedly send `Beacon` messages until the forwarder research phase expires:

- After a predefined period of time `BEACON_PERIOD`,

- Or, when a predefined number of `Reply`, `MAX_NB_REPLY`, messages is received.

When the node u receives a `Reply` message from any node v, it has access to the following parameters:

1. The residual energy of v.
2. The distance of v relative to the gateway.
3. The RSSI of the `Reply` message received from node v.

Depending on the **policy** (e.g. emphasis on energy, or lower latency, ...), u will select the most appropriate forwarder. In our experiments, we used a policy purely based on the distance of v coupled with the RSSI threshold.

Once the forwarder is selected, the node u transmits its packet. In ODYSSE, whenever no forwarder is found after `BEACON_PERIOD`, this period is extended (persistence strategy), unlike ASSORT [13] for instance where nodes return to the sleep state before proceeding to forwarder research. However, the persistence approach tends to minimize delays, and we believe that it is more efficient for dense networks.

The advantages of this design are the low storage capacity required and a light-weight selection procedure. Furthermore, ODYSSE, unlike for instance ORW [7] is loop free and guarantees packet unicity in the network. The design of the forwarder selection allows for a large spectrum of forwarder selection and forwarding strategies.

2.5 Duty Cycle

2.5.1 Principles

In ODYSSE, source nodes and gateway are always active while routers are duty cycled. The maximum duration of the active mode is fixed, while the sleep period follows a random uniform distribution. Ultimately, the duty cycling depends on the network traffic and indirectly on the density of the network.

The detailed functioning of a node, starting from the active state is as follows:

- If the node is idle (has no data packet to send), it waits for `Beacon` messages:
  - If after an active mode duration of `ACTIVE_PERIOD` no `Beacon` message is received, it returns to sleeping mode.
  - Otherwise, if the node actually replied to a `Beacon`, it waits for a `Data` message, and:
  - Then if the node does not receives a `Data` message after a period of duration `WAIT_DATA_PERIOD`, it returns to sleeping mode
  - Otherwise, it will forward the `Data` message as follows:
- If the node has data to send: it sends periodically `Beacon` messages, collect replies as described in section 2.4, until a forwarder has been successfully selected and then the `Data` packet has been successfully sent as unicast, as evidenced by receiving MAC layer acknowledgements. After that, the node has no data packet to sent and returns to sleep mode.
- **Sleep duration** (`sleep_period`) is a **random value** tuned following two application scenarios (see Section 2.5.2).

2.5.2 Sleep Duration

The duty cycle of nodes is critical because it directly impacts data transmissions, routing overhead and energy efficiency. In ODYSSE, nodes are unsynchronized, they randomly select a random sleep period in an interval:

$$[\texttt{MIN\_SLEEP\_PERIOD}, \alpha \times \texttt{ACTIVE\_PERIOD}] \quad (1)$$

Where the `ACTIVE_PERIOD` is the maximum active period of routers, and α is a constant. However, ODYSSE design carefully tunes this expression taking into account three application scenarios:

• `MED_ADAP`: (MultimEDia ADAPtive) This scenario considers a bulk data transfer of still images from the source.

In such setting, meeting delay constraints is possible with the following optimisation: router nodes adapt their duty cycle, by assuming existing traffic, and shortening their duty cycle. Consequently, the applied algorithm is as follows. When any router node transmits a packet, instead of turning back to the sleep state with a period given by formula 1, it rather sleeps for the minimum period `MIN_SLEEP_PERIOD` for a predefined number of times `SHORT_SLEEP_COUNT`. That is, before `SHORT_SLEEP_COUNT` packets transmitted, the node sleeps a minimum period of time. After that, it returns to the normal behavior. With this adaptive strategy, the network will face the congestion effect resulting from the bulk transfer. We will see that this method significantly reduces end-to-end delays without compromising the energy saving.

• `MED_N_ADAP`: (MultimEDia Non ADAPtive) This scenario is the same as the `MED_ADAP`, but nodes do not adapt their duty cycle, they just apply (1). This scenario is used as a reference for comparison.

• `INFR`: (INFRequent) This scenario represents a classical scenario in WSNs: infrequent event reporting which is characterized by low volume data generation by the source either periodic, regular, rare or exceptional. With this assumption, router nodes are allowed to keep their initial duty cycle scheme by applying formula 1 as a current packet is rarely a predictor for an additional incoming packet.

These scenarios are evaluated and compared experimentally (Section 4).

2.6 Example of Protocol Operation

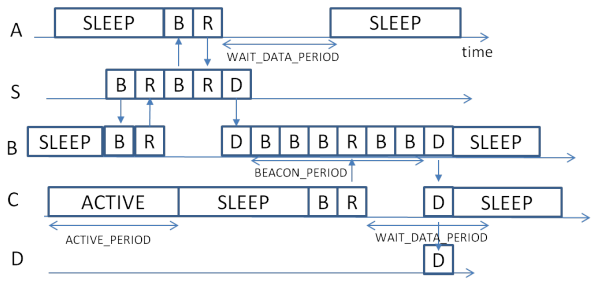


Figure 2: Illustrative diagram for data routing in ODYSSE.

Figure 2 illustrates the general functioning of ODYSSE. The source S has a data packet to send ('D'). It starts by transmitting beacons ('B'). We set the parameter `MAX_NB_REPLY` to 2, hence node S should wait for two `Reply` messages to send before selecting the forwarder. Node B

replies ('R') before node A because this latter was in the sleep state. Node B remains in the active state for a maximum period given by `WAIT_DATA_PERIOD`. At node A, this period expires with no data received, the node then returns to the sleep state. Receiving two Reply messages, the source S selects node B as a forwarder and transmits the data message. Similarly, node B starts sending beacons. When it wakes-up, the node C replies. However, as the `MAX_NB_REPLY` set to 2, node B waits the end of the `MAX_BEACON_PERIOD` to transmit the packet ('D') to node C. Node C finally forwards the data to the final destination D.

2.7 Packet Erasure Codes in ODYSSE

This section describes how ODYSSE is able to combat wireless transmissions losses. In our testbed, communication is assured by XBee modules which implement the MAC and physical layers of IEEE 802.15.4 in the 2.4 GHz channel. Direct Sequence Spread Spectrum (DSSS) is employed where a 4 bit symbol is mapped to a 32-chip pseudonoise-code, which already allows some (limited) error correction capabilities at the symbol level [41]. After symbol decoding/recovery, XBee modules will discard any packet with an incorrect Cyclic Redundancy Check (CRC), and pass the remaining packet to the microcontroller (e.g. our protocol). Conventional reliability improving approaches such as full data replication or on-demand retransmission are not suitable especially under high lossy channel as they impact delay and energy. Whereas, Forward Error Coding (FEC) is a good option. In addition, because ODYSSE operates at the routing layer, our focus is on the packet level rather than the bit level. Indeed, we consider, as in the RFC [30], a **packet erasure channel**, that is "a communication path where packets are either dropped or received. When a packet is received, it is assumed that it is not corrupted". Consequently, ODYSSE considers Erasure codes (EC) [26], which are based on **data redundancy**. Indeed, instead of sending k original packets, the **encoder** adds m **redundant** packets to recover the lost packets. When the number of the received packets is sufficiently high, the **decoder** decodes them and retrieves the original packets.

Different coding methods exist. In our work, we rely on the LDPC ("Low-density parity-check") codes [27]. LDPC code is a block code defined by a sparse parity-check matrix (that is, the majority of entries are zero), and is known to provide excellent decoding performances [28].

2.7.1 Preliminaries

The terminology is the same as the one from RFC 5170 [30]. Packets are supposed of equal size and are considered as vectors of $GF(2)$. We depart from some traditional presentations, as in our context the *symbols* are actually these *packets*. Hence obviously vectors of symbols can be represented as matrices. Consider a source node having k original packets, which can be considered as row vectors of a matrix, denoted S. The erasure code consists in generating more coded packets from the source packets, so that losses could be recovered. In our case, source packets are encoded with systematic codes: along with the k source packets, m additional **repair packets** are generated yielding $n = k + m$ packets. The matrix representing all packets (source and repair packets) is denoted P.

In general LDPC codes (systematic or or not systematic) are linear mappings: P is obtained as $P = G^T S$, where G is a **generator matrix**. By definition of LDPC, the code has one parity check matrix H, and the generator matrix must satisfy $HG^T = 0$ and hence $HP = 0$.

For decoding, the method is to recover the source packets from the received coded packets: because of the linearity, in principle, in suffice to solve the linear equations induced by $HP = 0$, where the unknown are the source packets while the constants are the received packets. This could be solved by Gaussian Elimination, but for LDPC codes, numerous, more efficient, algorithms exist [28] .

2.7.2 Practical Implementation with LDPC codes

In ODYSSE, the erasure correction works as follows:

1. As indicated, we use systematic coding, that is the original source packets S are sent uncoded: S is a submatrix of P. For simplicity of exposition, we assume in the following that the source packets follow the repair packets (in the matrix), hence the matrix P is composed of $m = n - k$ repair packets denoted C followed by k source packets S. Figure 3 illustrates the relation between parity check matrix and generated packets

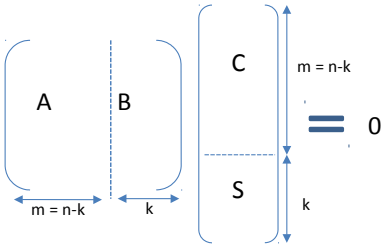


Figure 3: Relation between parity check matrix and generated packets.

2. Thus as $P = G^T S$, the last k columns of G will then be the identity matrix. Also, because $HG^T = 0$, it follows that B is equal to the identity. We have then: $C = -A^{-1}S$.

   Consequently, to generate the repair packets, ODYSSE pre-computes A^{-1}, the inverse of A, submatrix of H, the parity-check matrix[1].

   Binary LDPC codes are used, i.e. packets are vectors of $GF(2)$; addition corresponds to exclusive-or (XOR) operation on original packets S (and $-x = x$). In our case, we use the LDPC creation algorithms and source code from [29] to generate the matrices.

3. To generate the repair packets, one trivial solution would be to store all the original k packets and then apply $C = -A^{-1}S$. This requires a memory and storage overhead that usually does not fit low capacity sensors. Thus, ODYSSE stores instead the repair packets and updates them incrementally. Indeed, each repair packet is a linear combination of source packets. Thus, each time such a source packet S_j is generated ($j = 1 \dots k$), all the repair packets $C_i, (i = 1 \dots m)$ where S_j appears are updated: $C_i = C_i + S_j$ (and '+' corresponds to a XOR).

4. In routing, no distinction in made between repair packets or original packets, they are routed identically.

5. At the receiver side, the decoding is performed by solving starting from the relation $C = -A^{-1}S$. Every received repair packet thus can yield one linear equation involving source packets. The resulting set of linear equations may be naively solved for lost source packets. Zero, some or all of the lost source packets will be recovered, depending on the losses. Because the code is systematic, at least all source packets that are not lost, are available.

[1] LDPC code is sparse, however, the inverse is not necessarily sparse. It is worth taking this into consideration because the less the matrix is sparse, the more memory required to store its coefficients

3 Estimation of Forwarder Delay Performance

The forwarder selection is sender-initiated: a router collects replies from potential next-hop nodes, until it selects one. The critical question in opportunistic routing is how to choose the forwarder and how long a node should wait for potential forwarders. Waiting short time would decrease delays, but for applying QoS routing metrics (such as considering energy efficiency), a sufficient number of replies should be collected. In this section, we focus uniquely on the delay point of view, through a simplified theoretical model. We will determine the average waiting time of forwarders assuming general distribution of the wake-up intervals of the nodes: exponential and uniform distributions. Under these hypothesis, selecting the first forwarder is optimal, and estimate the propagation speed.

3.1 System Model and Notations

Consider a node that will route data to the first available forwarder. This forwarder is the first node that is closer to the destination. Our framework is essentially similar to [18] for instance (see their section 3.3.1). As in [18, 19], the wake up time and the position of the forwarders are independent. However a difference is that we will consider an asymptotic model where the destination is assumed to be at infinity, the network density is uniform, and the network infinite. We assume that the radio range is normalized and is considered to be equal to 1. We consider one step: a node at position $(0, 0)$ and the destination at $x = +\infty, y = 0$.

3.2 Average Progress when Taking the First Forwarder

The forwarder node will be the first one waking-up inside the half-disk of center $(0, 0)$ with radius 1 with $x \geq 0$ its position is a random variable X_1. Assuming that nodes are distributed uniformly and since wake-up time is independent from positions, the density of probability of the random variable X_1 is: $f_1(x) = \frac{4}{\pi}\sqrt{1-x^2}$. The average value of X_1 is:

$$\begin{aligned} E(X_1) &= \int_{t=0}^{t=1} t f_1(t) dt \\ &= \left[-\frac{4}{3\pi} \left(-x^2+1\right)^{\frac{3}{2}} \right]_0^1 \\ &= \frac{4}{3\pi} \approx 0.4244... \end{aligned}$$

This is the average amount of progress per hop towards the gateway when selecting the first forwarder (expressed in "radio range" units).

3.3 Discussion on Waiting for the k-first Replying Forwarders

As discussed in [19] for instance, the times, at which the k-first forwarders will reply, corresponds to the order statistics [17] of the random wake-up time of nodes. If a node has M possible forwarders, the time at which they will wake-up is given by the M random variables $Y_1, \ldots, Y_M$. The *sorted* sequence of these wake-up times is the *order statistics* and is denoted $Y_{(1)} \leq Y_{(2)} \leq ... \leq Y_{(M)}$. Now, we compute the average value of $Y_{(k)}$ in two specific, but largely applicable, cases:

• If each of the Y_i is exponentially distributed, then:

$$\begin{aligned} E(Y_{(k)}) &= \sum_{i=M-k+1}^{i=M} \frac{1}{i} \\ &= \frac{1}{M} + \frac{1}{M-1} + \ldots + \frac{1}{M-k+1} \end{aligned}$$

Furthermore, because even for dependent variables U and V, we have $E(U-V) = E(U) - E(V)$, therefore the mean delay between the k-th and $k+1$-th reply is:

$$E(Y_{(k+1)} - Y_{(k)}) = E(Y_{(k+1)}) - E(Y_{(k)}) = \frac{1}{M-k}$$

It results that for instance, $E(Y_{(2)}) - E(Y_{(1)}) = \frac{1}{M-1}$ and we have $E(Y_{(1)}) = \frac{1}{M}$. One fundamental result is that $E(Y_{(2)} - Y_{(1)}) > E(Y_{(1)})$, hence a node will wait on average (slightly) longer for the 2-nd reply than from the first. Intuitively, the difference comes from the fact that, after the first reply, there are only $M-1$ nodes which could reply. Naturally, if M is large, the mean delay between the first and the second reply is small.

Remark 1 *The distribution of $Y_{(k+1)} - Y_{(k)}$ is even given in [20], it is exponentially distributed as:* $\text{Exp}(\frac{1}{M-k})$.

• If each of the Y_i is uniformly distributed in [0,1], then the order statistic $Y_{(k)}$ has a beta distribution:

$$Y_{(k)} \sim \text{Beta}(k, n+1-k)$$

and then: $E(Y_{(k)}) = \frac{k}{n+1}$. Thus: $E(Y_{(1)}) = \frac{1}{n+1}$ and $E(Y_{(k+1)} - Y_{(k)}) = \frac{1}{n+1}$. This means that a node will wait on average as long for the 2-nd reply after having received the first, as it will wait for the 1-st reply.

3.4 Conclusions on k-first Forwarder Selection

These results mean that the general strategy of forwarding to the first node which replies is a viable default one:

- if transmission and then beacon initiation (forwarder research procedure) is considered to have zero delay (negligible).
- if the next-hop has itself as many (or more) potential next-hops as the current node.

Indeed the rationale is that:

- instead of having a node A waiting for a second reply from node C
- it can forward to the first node B that replies
- the node B that replied can send a beacon to find a next-hop of itself
- on average the node B will get a first reply from a node D at least as fast as the node A will get the reply from node C. And furthermore, on average (as the scheduling is independent from positions), D will be closer to the destination than C.

Note that nevertheless, a node might still improve on this strategy be considering the value $Y_{(1)}$ itself: when $Y_{(1)}$ happens to be larger than its expectation, $Y_{(2)} - Y_{(1)}$ might be more likely to be below its expectation. It should then consider the position of the replying node, to see if it might not be worthwhile to forward to it. However, due to previous remarks, it cannot expect this strategy to improve on the previous one, most of the time. Consequently, for our experiment, we will consider the heuristic of choosing the first forwarder.

3.5 Forwarder Selection and Throughput

When adaptative duty cycling (`MED_ADAP`) is used, the method will tend to make all the nodes awake, on the available paths between source and destination. If the metric is the total delay to achieve a bulk transfer, what would be the best forwarder selection algorithm? Ignoring as a first approximation the limiting effects of physical and MAC layers, and assuming for instance that one node may receive only one packet per unit-time, the question would be: what is the forwarder selection achieving max-flow[2]?

In ODYSSE, we did not provide an answer, and in this section, we give an intuition of why it is difficult to achieving it while being consistent with duty cycling.

A distributed max-flow algorithm could be implemented, for such as the variants of *push-relabel* algorithm reviews and proposed in [42] for instance. However, although push-relabel is one of the most natural method in a distributed setting, in our case of duty cycling, the lack of pre-defined routes and synchronous communication makes the problem significantly harder.

Assuming that the max-flow algorithm is not run prior a bulk transfer: if it is run online as the packets are flowing in the network, then a simple push-relabel algorithm cannot be used as is. In a push-relabel method, each node u (vertex) of the network (graph) is assigned a *height*, $h(u)$, and the flows are transferred only from nodes with higher height to nodes of lower heights.

However in traditional version, the height evolves dynamically with the algorithm until it ends (excess flowing back to the source). If we want to run it online, one permanent height assignment would be needed.

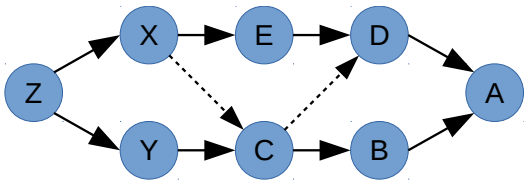


Figure 4: Max-flow labeling difficulties.

Figure 4 shows a counter-example of topology where no such assignment is possible. Assuming unit capacity link, max-flow (value=2) can only be achieved if the links $C \to D$ and $X \to C$. To prevent their use, while still using all the other, with a pure height method, one would need:

- $h(C) < h(D)$ (forbid this link), $h(E) > h(D)$ and $h(C) > h(B)$ (allow these links). Therefore: $h(E) > h(D) > h(C) > h(B)$
- $h(X) < h(C)$ (forbid this link), $h(X) > h(E)$ and $h(Y) > h(C)$ (allow these links). Therefore: $h(Y) > h(C) > h(X) > h(E)$

The two constraints are contradictory as the first one indicates that $h(E) > h(C)$, while the second that $h(C) > h(E)$.

[2]Notice that the max-flow is not in general achieved only by using shortest routing paths.

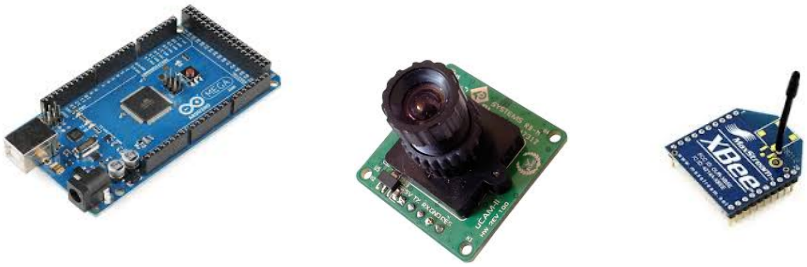
This difficulty of directly applying a base variant of push-relabel max-flow algorithm is the reason why ODYSSE does not attempt to maximize throughput by such methods.

4 Testbed Deployment and Experimentations

4.1 Testbed Hardware

45 Arduino nodes are deployed an area of 65×10 meters. Each node has two components: the ATmega2560 [31], a 8-bit microcontroller (256 KB of flash memory and 8 KB of SRAM) and the radio module XBee [32] (IEEE 802.15.4). The transmission power is set to 2 dBm. Source nodes are in addition equipped with TTL serial jpeg camera [34].

See Figure 5 for the hardware used.



(a) Arduino Atmega2560 (b) TTL serial jpeg camera (c) XBee S1

Figure 5: Hardware used in the testbed

The testbed is integrated in the existent large testbed FIT IoT-LAB [33], as described in [35, 43].

4.2 Integration of the Testbed in FIT IoT-LAB Platform

FIT IoT-LAB testbed [33] provides a very large scale infrastructure facility suitable for testing IoT protocols for constrained devices in . This infrastructure consists in six large-scale Internet of Things testbeds in France with over 2700 wireless sensor nodes in a variety of topologies and environments, and both fixed and mobile nodes. It is open for scientific and experimental usage. The platform can be accessed remotely through a web portal or using provided Linux commands. Integrating Arduino nodes within this testbed consists in linking each Arduino node to one FIT IoT-LAB node, called A8 node, via serial connection. Thus, the A8 node acts as a gateway for the Arduino nodes enabling us to remotely administrate the deployment (flash, debug, monitoring, etc).

4.3 Experiment Settings

The source node generates photos periodically (every 30 seconds). In experiments, we vary the value of α ($\alpha = 0$: no duty cycle is applied) and the data redundancy $\beta = m$, that is the number of repair packets sent by the source. For the `INFR` scenario, the source generates data packets representative of infrequent traffic, with a random inter-packet delay ranging from 5 to

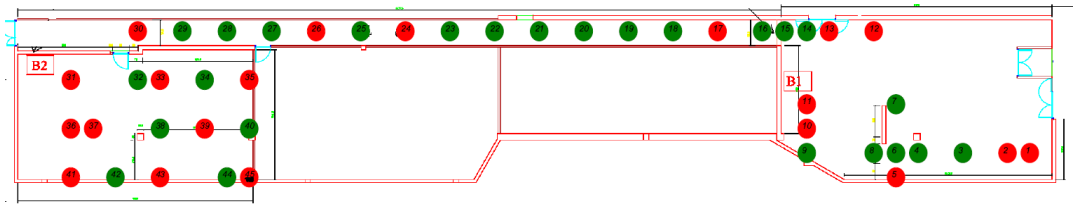


Figure 6: Real platform.

10 seconds. The other experiment parameters are summarized in Table 1. Results presented in this section are the average of 2 to 8 images.

Table 1: Experiment parameters

`ACTIVE_PERIOD`	0.2 s	`WAIT_DATA_RPERIOD`	3 s
`MIN_SLEEP_PERIOD`	0.05 s	`RSSI_THRESHOLD`	-83 dBm
`LEVEL_PERIOD`	8 s	`BEACON_PERIOD`	3 s
`MAX_NB_REPLY`	1	`SHORT_SLEEP_COUNT`	3

Our result analysis originates from detailed logs collected through the serial output of Arduino devices. Python tools retrieve, store, parse, and analyse the logs[3].

4.4 Topology

Figure 6 maps the actual physical placement of the nodes. In red are nodes equipped with cameras. To obtain an insight of the radio topology, a prior experiment ran a simple topology discovery algorithm based on a periodic broadcast of `Hello` messages. Thus each node is associated with its neighbors and the RSSI of received messages. Figure 7 depicts for each node (the x-axis), one dot for the RSSI (the y-axis) of one received `Hello` message of each neighbor. For clarity, only one RSSI per neighbor is presented in this Figure. The figure shows that node 1 has received 9 `Hello` messages with RSSI ranging from -88 dBm to -58 dBm. The density, measured as the average number of neighbors per node is about 16. Note that from routing point of view, this density does not represent the real average number of potential forwarders per node, as these neighbors do not necessarily ensure a geographic progress towards the final destination.

As highlighted in Section 2, in a preliminary phase, nodes start by computing their distance relative to the gateway. An example of the logical tree formed is given by Figure 8. Note that tree does not plot all neighboring links, nor all the possible parents (potential forwarders: the tree is actually a DAG). Almost all links have `link-metric` equal to 1. Nodes 12 and 26 have a link with low RSSI (below the threshold), hence a `link-metric` equal to 2. Unless explicitly mentioned, the source is the node with address 41 and the gateway is the node with address 6. Notice that the source node has the distance 6 from the gateway.

4.5 Overhead of ODYSSE

The main cost in term of energy efficiency, is closely related to the average number of beacons necessary to forward to one packet from one hop. Thus, we evaluate the overhead of ODYSSE

[3]collected logs represent a volume of 26 GBytes

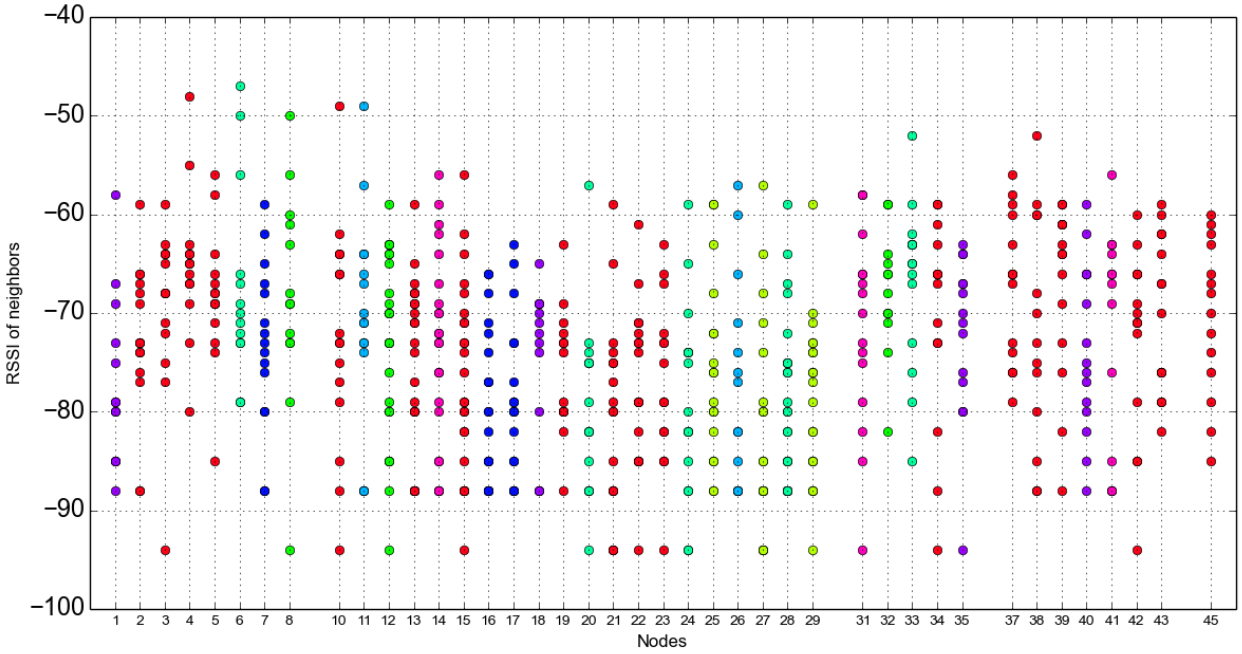


Figure 7: Radio topology.

in terms of number of beacons sent per data packet per node when varying α. To compute this average, only nodes having participated in routing (selected as next hop) are taken into account. Note that if all network nodes are taken into account, the plotted average would decrease. Also, only beacons that are sent for data packets that have reached the destination are taken into account.

As expected, results depicted in Figure 9 show that the overhead increases with α, and it is almost linear. However, the increase is more noticeable for `MED_N_ADAP` scenario, because in this scenario, no particular measure is applied to compensate the duty cycling of nodes. This result, actually, confirms the traditional trade-off concerning energy consumption between:

- Duty cycling: the energy consumption is inversely proportional to the duty cycle increase.
- Forwarder discovery: the energy consumption is proportional to the average time to find a forwarder, with is roughly proportional to duty cycle increase. When nodes sleep longer time, chances to find a forwarder are reduced and ODYSSE stays active and sends more beacons to discover one proper forwarder.

In `MED_N_ADAP` and `INFR`, on average, nodes are expected to have similar duty cycle for the same α. Hence what explains the overhead gap between them is the packet generation rate. Indeed, when data packets are sent less frequently (`INFR` scenario), the network is less congested. Hence, nodes find a forwarder more rapidly reducing the beaconing overhead. Similarly, `MED_ADAP` requires much less beacons than `MED_N_ADAP`. This proves that the duty cycle adaptation of `MED_ADAP`

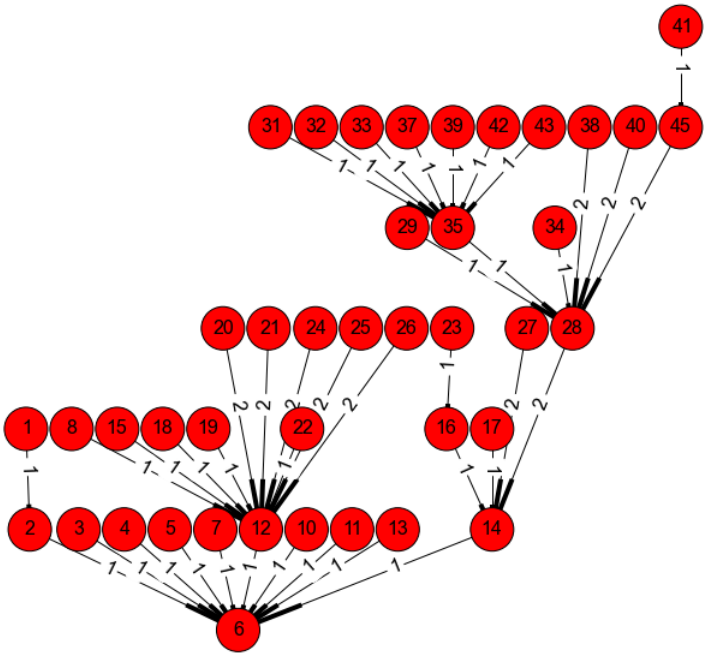


Figure 8: Example of a distance tree computed by nodes.

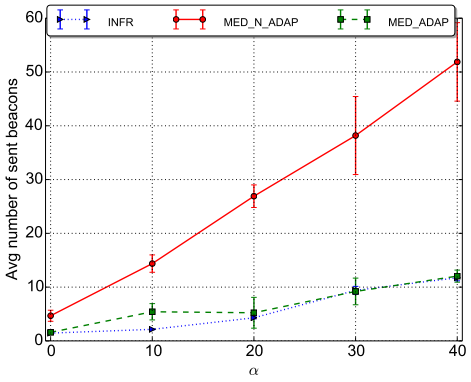
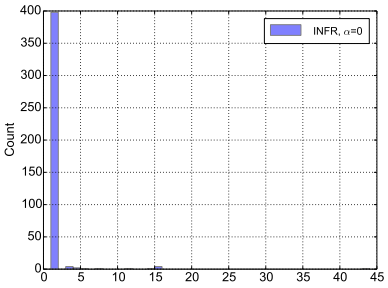


Figure 9: Average number of beacons sent per data packet per node.

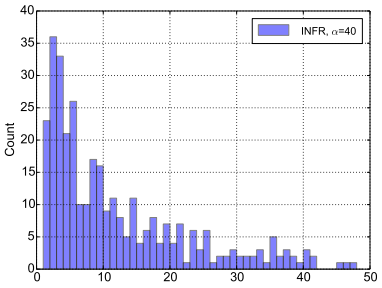
improves the forwarder availability during the bulk transfer duration and then accelerates their research.

Now, looking more in details, we can analyse the distribution itself of the number of beacons for a sample image (Figure 10). The immediate observation is that, with `INFR` and $\alpha = 0$, each node can immediately find a forwarder as in Figure 10(a), almost after sending one beacon.

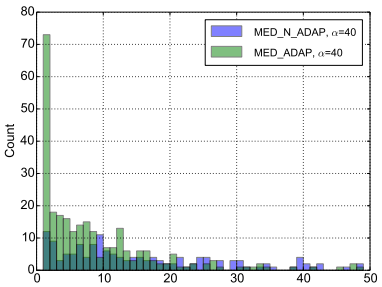
More interesting, is the result with low duty cycle (high α) in Figure 10(b): it appears that the distribution is more varied. The number of beacons is expected to be proportional to the waiting time given by the Beta distribution Beta$(1, n)$ previously presented (where n is the number of admissible forwarders of a node). Notice that the density function of Beta$(1, n)$ given



(a) `INFR`, $\alpha = 0$



(b) `INFR`, $\alpha = 40$



(c) `MED`, $\alpha = 40$

Figure 10: Distribution of the number of sent beacons.

by $f(x) \propto (1-x)^{n-1}$ visually corresponds to the general shape of the measured beacon density function.

Figure 10(c) shows that `MED_ADAP` scenario still properly illustrating its adaptiveness, since the most probable number of beacons necessary to find a forwarder is 1, in stark contrast with `MED_N_ADAP` scenario.

The lower efficiency of `MED_N_ADAP` compared to `INFR` can be noted (e.g. through the lower probability to find a forwarder with a low number of beacons, closer to 1). The explanation is as follows: in both scenarios, the nodes able to find forwarders faster are nodes with a larger number of neighbors. But, in `MED_N_ADAP` the network saturation is reached. In consequence, even if forwarders are found faster, forwarders themselves have to actually forward the packet to the next hop, before being available again: the benefits of having a large number of neighbors (favoring low delay) are thus naturally cancelled by congestion. This also explains the large gap between `MED_N_ADAP` and `INFR` in Figure 9.

4.6 Packet Inter-arrival Time

Figure 11 measures the average inter-arrival time of data packet at the gateway node as a function of α. It corresponds to the data reception rate at the gateway.

The average inter-arrival time of the `INFR` scenario is higher than the two other scenarios as the source generates packets with a random period ranging from 5 and 10 seconds. For the `INFR` and `MED_ADAP` scenarios, the inter-arrival time slightly increases with α. The `INFR` does not suffer from the network congestion. However, when α increases the reception rate at the gateway tends

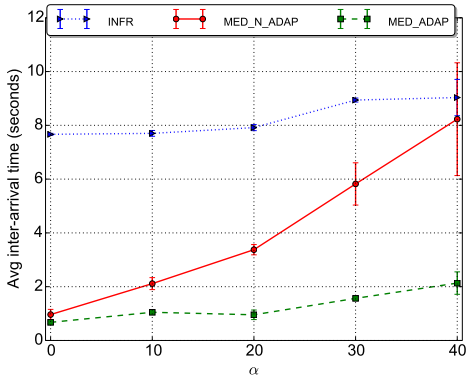


Figure 11: Average inter-arrival time of data packet.

to increase. Similarly, in the burst traffic scenario, the duty cycle adaptation of `MED_ADAP` is compensating the duty cycle effect of nodes, the network congestion however contributes to the reception rate increase in this scenario. The `MED_N_ADAP` has the highest reception rate, which is relatively highly sensitive to α. This fact is directly linked to the end-to-end delays distribution as explained in the next section.

4.7 Average End to End Delays

Figure 12 depicts the average end-to-end delay per data packet for each image as a function of α. For any given packet, this delay is measured as the time difference between reception time at the gateway and the transmission time at the source of this packet. Then, for each image, the end-to-end delay of a packet is the average value of end to end delay of all its packets.

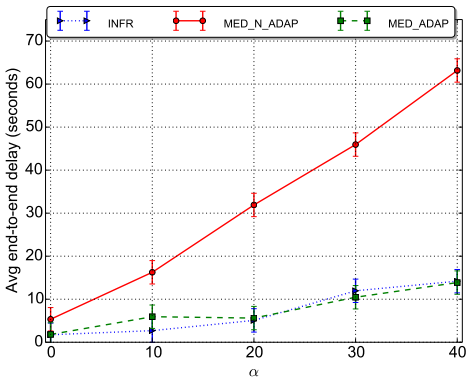


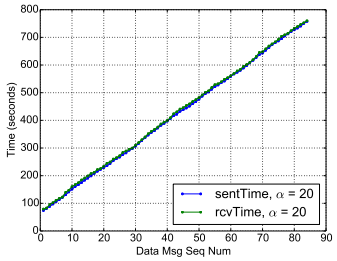
Figure 12: Average end to end delays per data packet.

As depicted in Figure 12, the end-to-end delays vary linearly with α. The scenarios `MED_ADAP` and `INFR` have low end-to-end delays that slightly increase with α. Hence, in these scenarios there is a good trade-off between the energy saving and the delay reduction. In `MED_N_ADAP`, we see that the delays are rapidly increasing with α. This is because routers are less available due to the duty

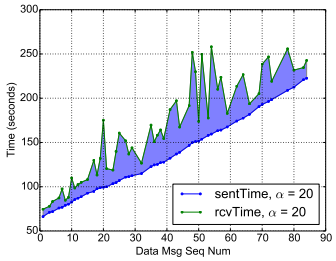
cycle and the network may reach congestion states because of the bulk transfer. These facts are alleviated by the duty cycle adaptation (in `MED_ADAP`) and the traffic regularisation (in `INFR`). While the traffic regularisation is dependant on the application, (not possible in multimedia scenarios in particular), the duty cycle adaptation is an effective method to reduce delays. Also, as we will see hereafter, this adaptability is of a less impact on the energy consumption.

Notice that the delay plots follow the beacon overhead plots as the beaconing process is basically a source of packet forwarding delays.

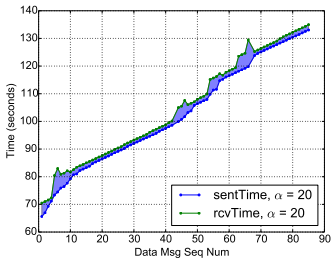
Delays are due to packet losses and to forwarders research time. A good visualisation to make the packet delivery more tangible is illustrated by Figure 13 presenting the times for sending and receiving packets of a sample image for $\alpha = 20$ for the three evaluated scenarios. Obviously, the colored area explains the source of end-to-end delays. For instance, in `MED_N_ADAP` scenario, plots have high amplitude, spikes correspond to some routers that are blocked because of either network congestion or duty cycle. As explained above, `MED_ADAP` and `INFR` are less impacted by these phenomena, no spikes for `INFR` scenario and some low amplitude spikes for `MED_ADAP` scenario.



(a) $\alpha = 20$, `INFR`



(b) $\alpha = 20$, `MED_N_ADAP`



(c) $\alpha = 20$, `MED_ADAP`

Figure 13: Time of sending and receiving of packets of a sample photo.

4.8 Packet Delivery

Figure 14 depicts the average packet delivery ratio (PDR). This PDR is computed as the ratio of successful data packet transmissions by any node in the network. Notice that in our experiments, the error correction is enabled, and thus, recovered packets following this error correction are considered as successful transmissions. This figure gives an idea about radio conditions and packet losses in the network. For the three studied scenarios, the PDR ranges from 90% and 98% on average for different values of α. In theory, the PDR is not directly impacted by the sleep

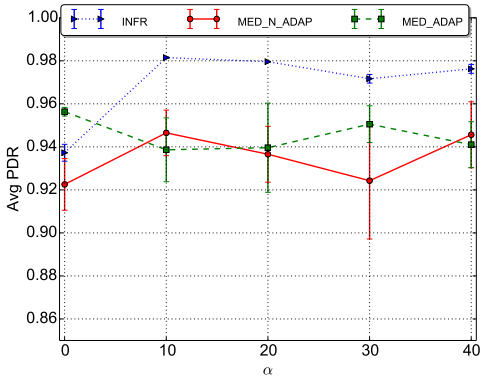


Figure 14: Average PDR of data packet.

period of nodes, and hence α. However, one would think that when nodes sleep longer time, more beacons are generated (as highlighted in Figure 9 for the `MED_N_ADAP` scenario), and hence more interferences are likely to occur. This hypothesis is not confirmed in our experiments as the PDR plot is not strictly monotonic for this scenario. Globally, from our log files analysis, we noticed that the packet losses are due to packet collisions or in few cases to the research forwarder failures. These aspects are less present for the `INFR` scenario, thus its high PDR values.

4.9 Duty Cycle

Figure 15 depicts the average duty cycle per node, defined as the percentage of the average sleep time of this node during its lifetime. The average values depicted in Figure 15 are the average on all nodes for all images transmitted. Notice that when router nodes are in the active state, they are either (1) waiting for beacons, (2) searching forwarders, (3) or waiting for data packets after having transmitted a `Reply` message. All these states have a maximum duration after which nodes return to the sleep state, except after a research forwarder failure that should be extended.

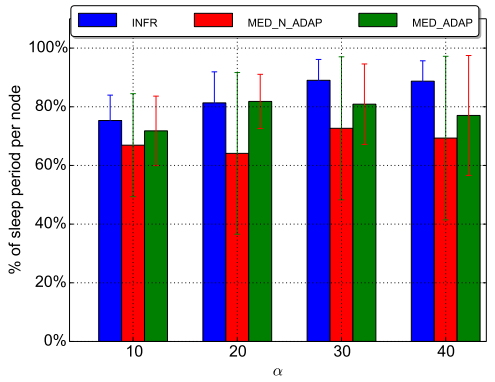


Figure 15: Average duty cycle per node.

As explained in Section 2.5, the sleep duration is given by a random value in a prede-

fined interval: [MIN_SLEEP_PERIOD, $\alpha \times$ ACTIVE_PERIOD]. Given the parameters setting, in a given cycle, for $\alpha = 10$ for instance, any node would sleep an average period approximated by random$(50, 200 * 10) \simeq 1025$ ms, in MED_N_ADAP. Which leads to a sleep ratio equal to: $\frac{1025*100}{1225} = 83\%$. Experiments yield an average value $\simeq$ 70%. A duty cycle of 70% means an energy gain of the same value. Notice also that many nodes reach a duty cycle of 90%.

Another observation form Figure 15, is that, as expected, INFR scenario ensures the highest values of the duty cycle. The average duty cycle increases with α for INFR scenario, while it is almost insensitive to α for MED_N_ADAP and MED_ADAP. However, this average hides the differences between nodes although they do not participate in routing equally. Hence, it is worthy to evaluate the duty cycle of each node as depicted in Figure 16 for $\alpha = 10$ and $\alpha = 30$.

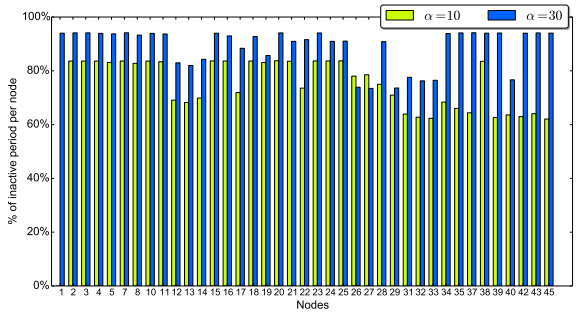
From this figure, we notice that the duty cycle depends on α; as it determines the inactive period selected by any node. Conceptually, high values of α mean that nodes are allowed to be in the inactive state for longer time. For this reason, $\alpha = 30$ is globally ensuring higher duty cycle than $\alpha = 10$. However, increasing α at any node, means that less neighbors of this node will be awake. Thus, this node spends longer time searching for forwarders which decreases its duty cycle and increases its energy consumption. This the reasoning applies for all nodes. Consequently, less values of α would be more energy effective under high traffic conditions. This is what we see in MED_N_ADAP scenario: $\alpha = 10$ exceeds $\alpha = 30$ for some nodes. In contrast, in INFR and MED_ADAP scenarios, the duty cycle increases with α almost for all nodes. This means that regulating traffic injection (INFR scenario) and tuning the duty cycle (MED_ADAP scenario) allow the network to take advantage from high sleep durations setting. To conclude, the inactive period is essential for energy saving, but adapting it to the network state is crucial.

Another observation from these figures is that nodes have heterogeneous duty cycle; For instance nodes 30, 31, 34, etc are near the source (see the real testbed topology in [43]). When such a node becomes active, its chances to receive a beacon are high as this source is transmitting packets, and hence beacons, continuously (MED_ADAP or MED_N_ADAP). In contrast, when a router which is far from the source wakes up, the chances that one of its neighbors, which are routers, is sending a beacon are smaller than if it was next to the source. Thus, nodes close to the source have low duty cycle compared to other nodes despite their relatively low activity (see Figure 17). That said, Figure 16(a) proves that ODYSSE enables all nodes to have a good duty cycle. Also, from Figure 17, we see that even nodes with high activity reach good duty cycle. For instance, node 28 duty cycle is 60% despite it forwards $\simeq 60\%$ of the data traffic (MED_ADAP). Also, there are nodes that do not appear in Figure 17, their duty cycle is almost 90% for $\alpha = 30$ (nodes 1, 2, ...).

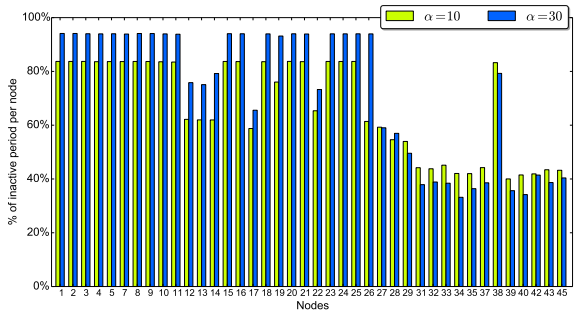
4.10 Evolution of the Number of Packets in the Network

Figure 18 depicts the evolution of the number of packets in the network, that is the total number of packets that nodes are buffering for transmissions.

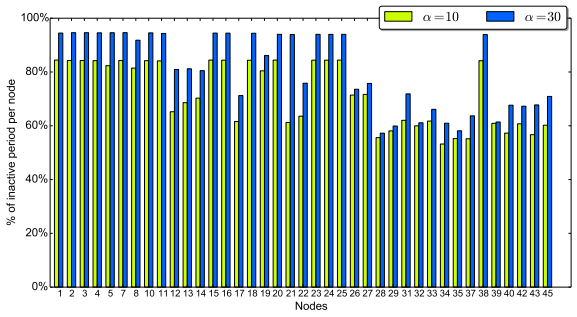
We compare two values of α, 0 and 40. In the INFR scenario, the number of packets in the network oscillates very much; as the packets generation rate is low, their total number rapidly decreases in the network (maximum number is equal to 4 packets distributed over all nodes). The plot with $\alpha = 40$ reaches higher amplitudes and higher width than the plot with $\alpha = 0$ in all scenarios. The difference in amplitude is due to the network congestion and to the duty cycle. However, the number of packets is the network oscillates less in MED_N_ADAP scenario and it reaches its maximum values before starting to decrease. This means, as highlighted above, that some packets are more blocked in this scenario than in MED_ADAP scenario. Also, higher values of α increase the packets sojourn time in the network leading to higher end-to-end delays; which is shown by the larger width of the plots for $\alpha = 40$.



(a) `INFR` scenario.



(b) `M_N_ADAP` scenario.

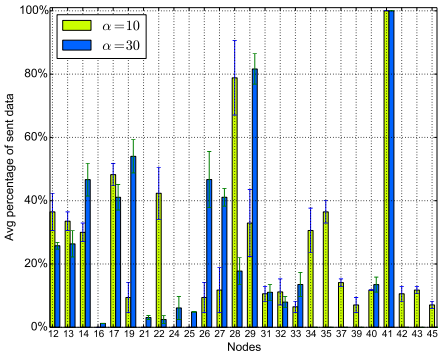


(c) `M_ADAP` scenario.

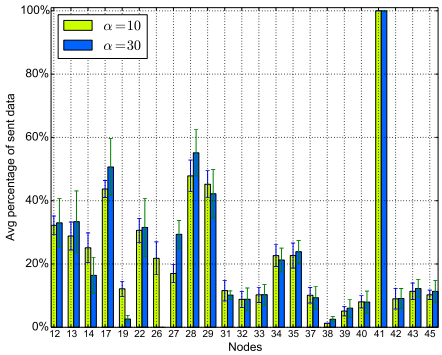
Figure 16: Individual duty cycle.

4.11 Error Correction

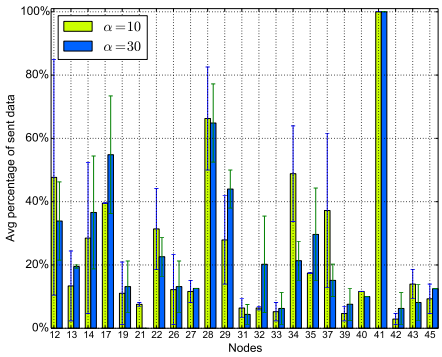
An important feature in ODYSSE is its integration an error correction module based on LDPC codes. The usage of such a code enables us to decode original packets which are sent by the source and lost in the network. Indeed, as explained in Section 2.7, in addition to the original packets (*uncoded packets*), the source node transmits redundant (repair) packets (*coded packets*) which are linear combinations of original packets. In erasure codes, the coding rate defined by: $\frac{\text{The number of uncoded packets}}{\text{The number of uncoded packets+the number of coded packets}}$ is a crucial parameter as it determines the redundancy introduced in the network. Thus, we evaluate the impact of the parameter



(a) `INFR` scenario



(b) `MED_N_ADAP` scenario



(c) `MED_ADAP` scenario

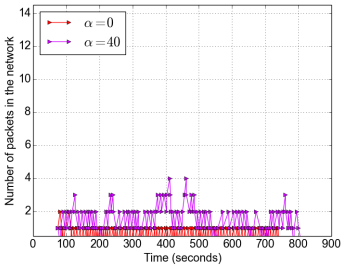
Figure 17: Average percentage of sent data packet per node.

β = The number of coded packets in this section. We base our experiments on the `MED_N_ADAP` scenario as it is a trivial solution for a challenging WSN scenario. α is set to 10.

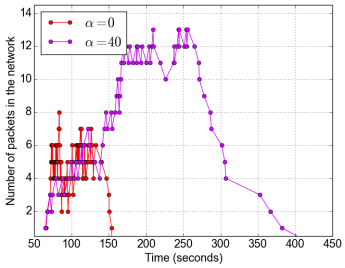
Figure 19 depicts the average number of sent packets, the average number of received packets and the average number of the decoded packets (total number of packets retrieved at the gateway that will be reassembled to re-constitute the image) for different redundancy levels (0 means no redundant packets are sent). Results are the average of 8 images of the same size.

First, Figure 19 highlights the added value of the LDPC codes. Without coding, decoded packets are evidently just those received by the gateway. Otherwise, when coded packets are sent, the gateway decodes more than the uncoded packets that it receives: this is the difference between 'uncoded received' and 'decoded' bars in Figure 19. In all cases, the number of decoded packets is higher than the number of received packets. This difference highlights the importance of the coding integrated in ODYSSE. For image transmissions for instance, losing some jpeg packets may prevent its decoding; the image cannot be displayed. Second, we notice that, the higher β is, the higher the number of decoded packets is: when $\beta \geq 30$, 100% of lost packets are recovered. Figure 20 which depicts the percentage of recovered packets among lost packets, proves this result.

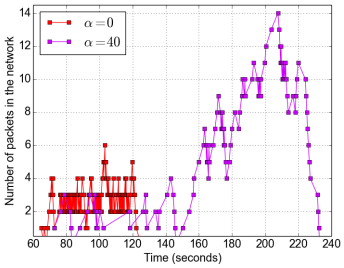
However, from our experience, we can argue that in implementing the LDPC codes, the



(a) `INFR` scenario.



(b) `MED_N_ADAP` scenario.



(c) `MED_ADAP` scenario.

Figure 18: Evolution of the number of packets in the network.

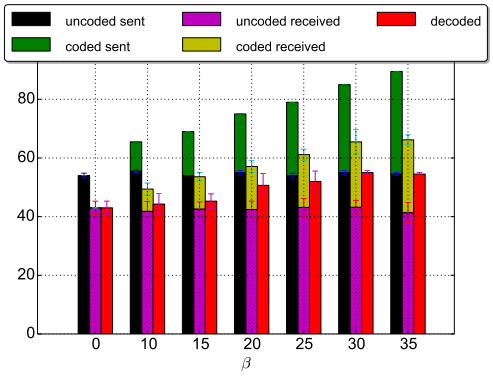


Figure 19: Decoding performance.

memory space of sensors is very stringent, as it requires to buffer coded packets and store the LDPC code. For our case, despite the optimized implementation of LDPC code as discussed in Section 2.7, we were limited in the maximum value of β. Our experience witnesses the importance of the optimized erasure codes.

We now evaluate the cost of the erasure codes in ODYSSE in terms of delays and duty cycle. Figure 21 shows that the end-to-end delays slightly varies with β. Recall that coded packets are transmitted following the same routing procedure and time frequency as uncoded packets. Thus, coded packets do not create a particular congestion phenomena in the network compared

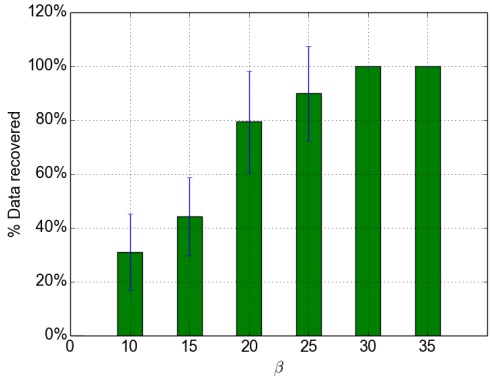


Figure 20: Percentage of data packets recovered.

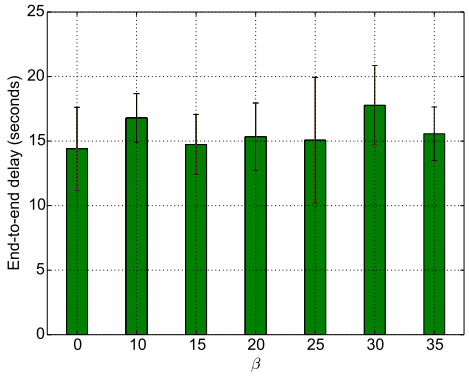


Figure 21: Average end to end delays per packet per data redundancy.

to uncoded packets and hence do not necessarily contribute to high variations of the average end-to-end delays per packet.

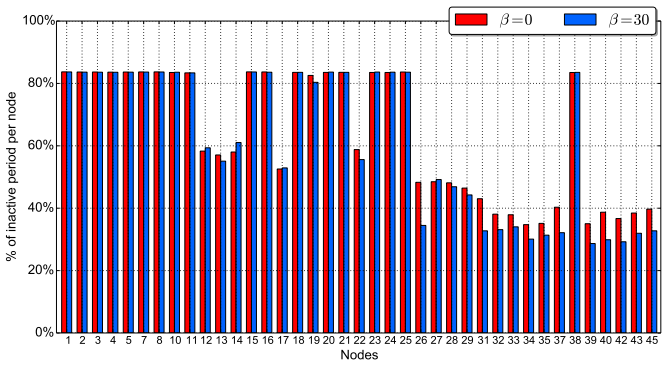


Figure 22: Average duty cycle per node as a function of β.

Figure 22 plots the duty cycle of nodes for $\beta = 0$ and $\beta = 30$. The usage of the erasure code inevitably decreases the duty cycle as nodes have to transmit coded packets in addition to uncoded packets. However, this increase is small and mainly visible at nodes close to the source; the overall energy consumption of the network is almost constant. Notice that in ODYSSE, the coded packets computation is done only at the source, which reduces the computational energy consumption. As a reference, our findings are coherent with those in [21] which claims that the erasure codes have notable reliability improvement at a small energy and delay cost.

5 Related Work

5.1 Opportunistic duty cycle based Routing in WSNs

Many works [7, 9, 13] have applied the opportunistic scheme to save energy in WSNs. In [7], ORW applies asynchronous low duty cycle and selects the potential forwarders according to the expected number of wakeups until a packet has reached its intended destination (EDC). ORW proposes then a method to select one forwarder, but packet duplication still be possible. In [8], authors proposes ORIA protocol which is an improved version of ORW supporting in-network aggregation. The protocols proposed in [7] and [8] require the exchange of duty cycle information to update the EDC metric. ODYSSE avoids this overhead by proposing a light-weight overhead forwarder selection method. In [9], the residual energy is used as a metric to select packet forwarders which contributes to balance energy consumption among all nodes. Similarly, ODYSSE uses, among others, the residual energy to select the forwarder, but in addition ODYSSE relies on the duty cycle to avoid the energy wasted in the idle state. In ASSORT [13], whenever a node has data to transmit, it broadcasts beacons to collect potential forwarders. If no acknowledgement is received, the node retries to find a forwarder after a sleep period. This waiting time increases data delivery delays. ODYSSE avoids this delay by a persistence strategy where nodes still active until the packet is forwarded. Besides, authors design a model to determine the wake-up rate and the awake period to apply the duty cycle based on the energy consumed in the probing and waking states. In ASSORT, the routing metric is based on: the residual energy, the link reliability, and sleep-wake schedules. RI-MAC [14] is a receiver-initiated MAC protocol. When any node turns on its radio, it transmits a short beacon frame to announce that it is ready to receive data. To avoid collisions, the receiver informs its neighbors about its collision window, that is the period that must be used to communicate with it. In the same category of receiver-initiated MAC protocols, CD-MAC [15] has been proposed more recently. In CD-MAC, once a sender receives a probe from a receiver, it acknowledges it. Then, the receiver will poll one after the other each of the potential forwarders having sent such an acknowledgement. The proposed scheme has been implemented in TinyOs on TelosB motes. Unlike RI-MAC, B-MAC [16] is sender-initiated. Each node willing to send data, transmits a "wakeup signal", called a preamble which lasts longer than the receiver sleep interval. Each node periodically wakes up to check if there is any preamble, in which case it remains active to receive possible incoming packets.

5.2 Error Correction in WMSNs

In wireless network communications, faulty equipment, congested routers or other problems can lead to a partial or complete loss of a packet. Conventional reliability improving approaches such as full data replication or on-demand retransmission are too expensive or even not possible due to very strict energy constraints and asymmetric channels. Erasure Codes (EC) allow to enhance the reliability of data transmissions by transmitting redundant data [21, 23, 24, 25].

An Erasure Code is a forward error correction code that, operating in the application layer [22, 30], assumes that a packet is either received without errors or completely dropped. ECs enhance data transmission reliability by introducing redundancy, without the overhead of strict replication. ECs consist of an encoding and a decoding algorithm. The former one extends a group of k packets to n packets by generating $m = n - k$ redundant packets, where $n > k$. Each subset of the n packets containing at least k' packets is sufficient to successfully decode the original data, where $k \leq k'$. The code rate $r = \frac{k}{n}$ describes the overhead in terms of redundant packets.

One major concern for EC mechanisms is the complexity of the mechanism itself. For instance, in [21], authors investigate the trade-off between reliability achieved and the energy and delay overhead for different EC techniques. They proved that ECs improve the communication reliability considerably with a small delays increase. In [23], authors conducted a theoretical study and real experiments on solar power sensors to dynamically adjust the redundancy of erasure codes regarding the energy level of sensors. The objective is that the end-to-end packet delivery probability is maximized and the network lifetime is not affected. The paper [24] focuses on body WSNs to investigate the decoding power consumption of LDPC codes [27]. Authors propose a scheme to determine the number of iterations in this code allowing sensors to save decoding energy while reaching a specific bit error rate.

In WMSNs, reliability is as a major concern as in WSNs. A lost packet may dramatically degrade video or still image quality. In [25], authors analysed the impact of retransmission and wireless application-layer redundancy by using packet arrival probability and average energy consumption. They concluded that using the erasure code is more reliable and energy efficient than retransmission when the packet loss probability is low, but the performance of the erasure code deteriorates when high packet loss conditions occur. Similarly, authors in [26] evaluate the performance of conventional error control schemes (EC, ARQ, etc) in WMSNs as a function of the bit error rate. They find that the best strategy is to use cross-layer scheme in low bit error rates and link-layer hybrid schemes in high bit error rates. In our work, to face transmission unreliability, we use the source coding based on the known LDPC codes.

6 Conclusion

This paper proposes ODYSSE, a novel design tailored for duty cycled WSNs networks to achieve energy efficient data forwarding without compromising end to end delays. ODYSSE is also characterized by a duty cycle setting which is adaptable to different application scenarios: bulk data transfer with/without adaptive duty cycle and infrequent data transfer. To validate our proposal, we conducted extensive experiments with a real testbed based on Arduino platform. Results show that adapting the duty cycle to network traffic conditions is essential, even usually more important that the sleep period itself. The application of the source coding in ODYSSE improves reliability at the expense of a low energy cost. We argue that, to maximize the gain from source coding (through higher code rate), an optimized implementation of these codes is mandatory. This paper provides a good reference for WSNs studies as it reveals the behavior of sensors under real wireless conditions and real application. Also, our experience witnesses the importance of wireless protocols parameters that should dynamically adapt to the environment.

ACKNOWLEDGEMENT

This work has been supported by the Celtic plus project TILAS [36].

References

[1] W. Ye, J. Heidemann, and D. Estrin, *An energy-efficient mac protocol for wireless sensor networks*, in IEEE INFOCOM, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, vol. 3, 2002.

[2] J. Polastre, J. Hill, and D. Culler, *Versatile low power media access for wireless sensor networks*, in Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys'04. New York, NY, USA: ACM, 2004, pp. 95-107.

[3] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*, John Wiley & Sons, 2005

[4] S. Biswas and R. Morris, *Exor: Opportunistic multi-hop routing for wireless networks*, SIGCOMM'05, vol. 35, no. 4, pp. 133-144, Aug. 2005.

[5] C. Szymon, J. Michael, K. Sachin, K. Dina, *Trading Structure for Randomness in Wireless Opportunistic Routing*, Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, Kyoto, Japan, 2007,

[6] X. Mao, S. Tang, X. Xu, X.Y. Li, H. Ma, *Energy-Efficient Opportunistic Routing in Wireless Sensor Networks*, IEEE Transactions on parallel and distributed systems, vol. 22, no. 11, November 2011.

[7] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, *Low power, low delay: opportunistic routing meets duty cycling*, in Proceedings of the 11th international conference on Information Processing in Sensor Networks, IPSN'12. New York, NY, USA.

[8] J. So and H. Byun, *Opportunistic routing with in-network aggregation for asynchronous duty-cycled wireless sensor networks*, Wireless Networks, pp. 1-14, 2013.

[9] M. Chien-Chun Hung1, K. Ching-Ju Lin, C.F Chou, and C.C. Hsu, *EFFORT: Energy-efficient opportunistic routing technology in wireless sensor networks*, Wireless communications and mobile computing, June 2013.

[10] I. F. Akyildiz, T. Melodia, K.R. Chowdhury, *A survey on wireless multimedia sensor networks*, Computer network journals, October 2006.

[11] I.T. Almalkawi, M. Guerrero Zapata, J. N. Al-Karaki, J. Morillo-Pozo, *Wireless Multimedia Sensor Networks: Current Trends and Future Directions* , Sensors 2010, Volume 10, Issue 7.

[12] M. Abazeed, N. Faisal, S. Zubair, and A. Ali, *Routing Protocols for Wireless Multimedia Sensor Network: A Survey*, Journal of Sensors, vol 2013, 2013.

[13] C-C. Hsu, M-S. Kuo, S-C Wang, C-F. Chou, *Joint Design of Asynchronous Sleep-Wake Scheduling and Opportunistic Routing in Wireless Sensor Networks*, in Computers, IEEE Transactions, vol.63, no.7, pp.1840-1846, July 2014.

[14] Y. Sun, O. Gurewitz, and D. B. Johnson, *RI-MAC: A Receiver-Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks*, in 6th ACM Conference on Embedded Network Sensor Systems. New York, NY, USA: ACM, 2008, pp. 1-14.

[15] D. Liu, X. Wu, Z. Cao, M. Liu, Y. Li, M. Hou, *CD-MAC: A contention detectable MAC for low duty-cycled wireless sensor networks*, in Sensing, Communication, and Networking, SECON 2015.

[16] J. Polastre, J. Hill, and D. Culler, *Versatile Low Power Media Access for Wireless Sensor Networks*, in 2nd International Conference on Embedded Networked Sensor Systems. ACM, 2004, pp. 95-107.

[17] H. A. David, H. N. Nagaraja, *Order Statistics* (Wiley Series in Probability and Statistics). Wiley-Interscience, August 2003.

[18] L. Sha, F. Kai-Wei, S. Prasun, *CMAC: An Energy-efficient MAC Layer Protocol Using Convergent Packet Forwarding for Wireless Sensor Networks*, ACM ACM Transactions on Sensor Networks (TOSN), vol. 5, no. 4, November 2009.

[19] N. kolar-purushothama, K. Anurag, *Tunable Locally-Optimal Geographical Forwarding in Wireless Sensor Networks with Sleep-Wake Cycling Nodes*, IEEE INFOCOM 2010.

[20] Timo Kosk, "*Order Statistics for Independent Exponential Variables*", notes of "Computer intensive methods in mathematical statistics", 2015

[21] M. Mutschlechner, B. Li, R. Kapitza, F.alko Dressler, *Using Erasure Codes to Overcome Reliability Issues in Energy-Constrained Sensor Networks*, 11th Wireless On-demand Network Systems and Services (WONS), April 2014.

[22] V. Roca, *Codes AL-FEC pour le canal aéffacements: codes LDPC-Staircase et Raptor*, `https://ensiwiki.ensimag.fr/images/8/8c/4MMCSR-VR-sceance3_al_fec.pdf`

[23] Y. Yang, L. Su, Y. Gao, and T. Abdelzaher, *SolarCode: utilizing erasure codes for reliable data delivery in solar-powered wireless sensor networks*, in Proc. IEEE INFOCOM'10 (mini-conference), Mar. 2010.

[24] N. Javaid,, O. Rehman, N. Alrajeh, Z. A. Khan, B. Manzoor, S. Ahmed, *AID: An Energy Efficient Decoding Scheme for LDPC Codes in Wireless Body Area Sensor Networks* , 2013 International Workshop on Communications and Sensor Networks (ComSense-2013).

[25] M. van der Schaar, S. Krishnamachari, S. Choi, X. Xu, *Adaptive Cross-Layer Protection Strategies for Robust Scalable Video Transmission Over 802.11 WLANs*, IEEE Journal on Selected Areas in Communications, vol. 21, no. 10, December 2003.

[26] M. Y. Naderi, H.R. Rabiee, M. Khansari, M. Salehi, *Error control for multimedia communications in wireless sensor networks: A comparative performance analysis*, Ad Hoc Networks. (2012).

[27] `https://en.wikipedia.org/wiki/Low-density_parity-check_code`

[28] M. Cunche, V. Roca, *Improving the Decoding of LDPC Codes for the Packet Erasure Channel with a Hybrid Zyablov Iterative Decoding/Gaussian Elimination Scheme*, Research Report 2008, pp.19. `https://hal.inria.fr/inria-00263682v1/document`

[29] `http://radfordneal.github.io/LDPC-codes/index.html`

[30] `https://tools.ietf.org/html/rfc5170`

[31] Arduino: `https://www.arduino.cc/en/Main/ArduinoBoardMega2560`

[32] Xbee: `http://www.digi.com/lp/xbee`

[33] FIT IoT-LAB: `https://www.iot-lab.info/`

[34] TTL camera: `https://www.sparkfun.com/products/retired/10061`

[35] ODYSSE: `http://odysse-upec.github.io`

[36] TILAS project: `http://www.tilas.eu/`

[37] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, *6TiSCH: deterministic IP-enabled industrial internet (of things)*, Communications Magazine, IEEE , vol.52, no.12, pp.36-41, December 2014 doi: 10.1109/MCOM.2014.6979984

[38] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, *A high throughput path metric for multi-hop wireless routing*, Wirel. Netw., vol. 11, no. 4, pp. 419-434, 2005.

[39] T. Watteyne, C. Adjih, and X. Vilajosana, *Lessons Learned from Large-scale Dense IEEE802.15.4 Connectivity Traces*, in IEEE International Conference on Automation Science and Engineering (CASE), Gothenburg, Sweden, 24-28 August 2015.

[40] C.E. Perkins, P. Bhagwat, *Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers*, ACM SIGCOMM '94), New York, NY, USA.

[41] K. Wu, H. Tan, H.L. Ngan, Y. Liu, L.M. Ni, *Chip error pattern analysis in IEEE 802.15.4*, Mobile Computing, IEEE Transactions on, 11(4), pp.543-552.

[42] Homayounnejad, S. and Bagheri, A. "*An efficient distributed max-flow algorithm for Wireless Sensor Networks*", Journal of Network and Computer Applications, 54, pp.20-32, Apr. 2015

[43] I. Amdouni, C. Adjih, N. AitSaadi & P. Muhlethaler, *ODYSSE: A Routing Protocol for Wireless Sensor Networks*, Research Report.

ANNEX

Figure 23 illustrates the flowchart of ODYSSE.

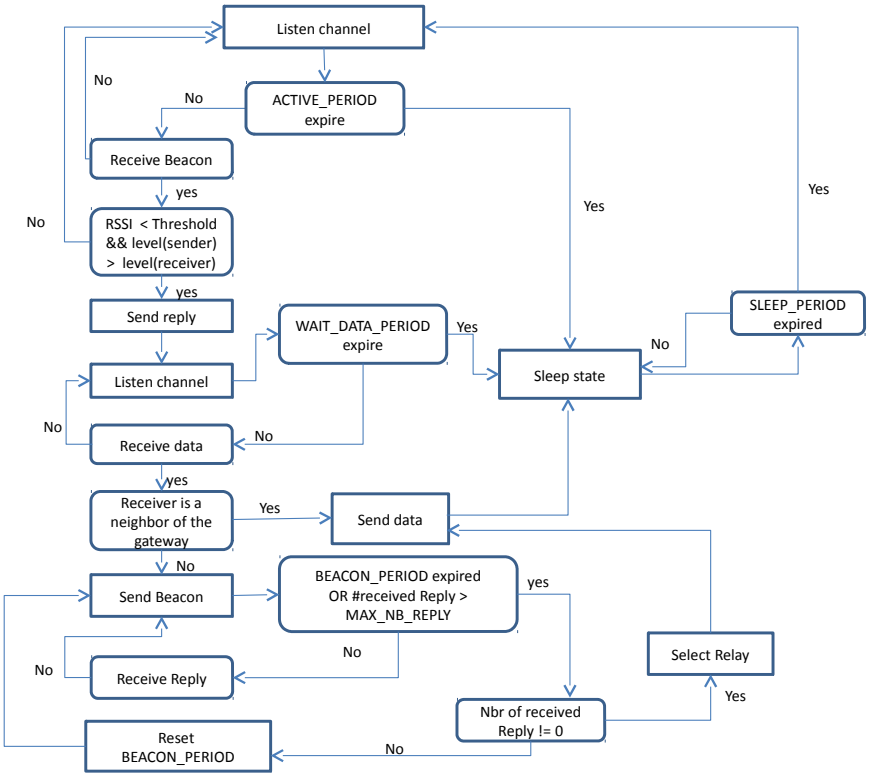
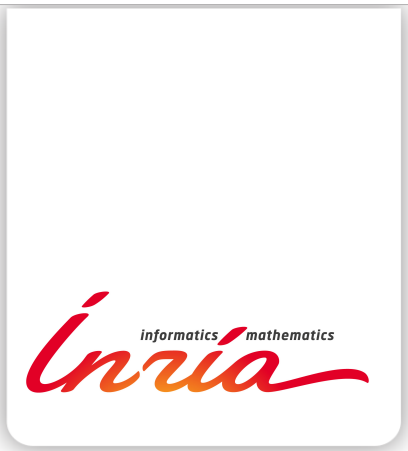


Figure 23: The flowchart of router nodes in ODYSSE.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399